RN의 위기의식으로 만들어진 JSI

JSI(javascript interface)는 react native의 New Architecture 이다.

기존의 리액트 네이티브는 자바스크립트와 네이티브 모듈 간 통신에 있어 브리지(Bridge)에서 과부하가 자주 걸린다.

리액트 네이티브 코드에 모듈을 추가 확장할수록 통신수가 증가하면서 과부하가 발생하게 됩니다.

이는 브리지의 특성상 어쩔수 없는 현상입니다.

그렇다면 기존의 리액트 네이티브 아키텍처는 어떻게 구성되었고, 이러한 단점이 발생할까?

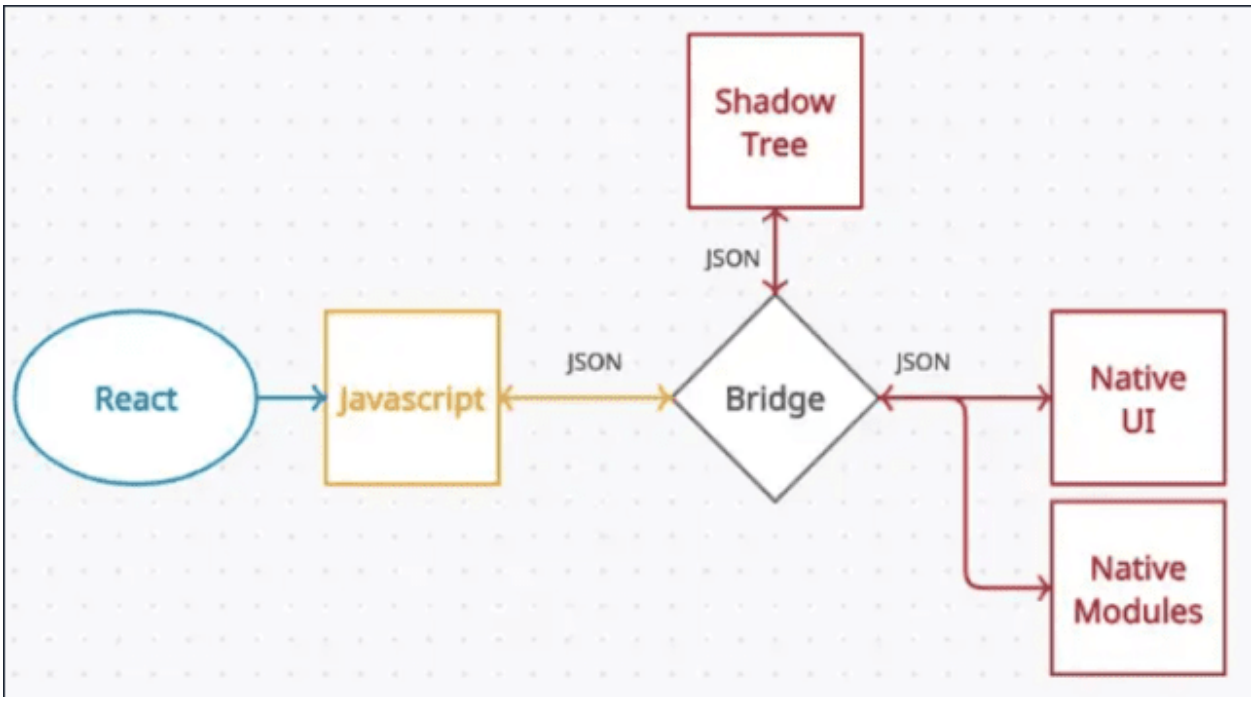
먼저, 기존의 리액트 네이티브는 3가지의 쓰레드로 구성되어있습니다.

- UI thread
- Shadow thread
- JS thread

세 개의 쓰레드 간의 통신은 직접적으로 통신되는것이 아니라, 브리지(Bridge)를 통해 이루어지는데, 이 브리지를 통할 때 데이터가 JSON 형태로 이동되는데, 시리얼라이제이션(serialisation)과 디시리얼라이제이션(deserialisation) 같은 무거운 작업을 통해 통신이 이루어집니다.

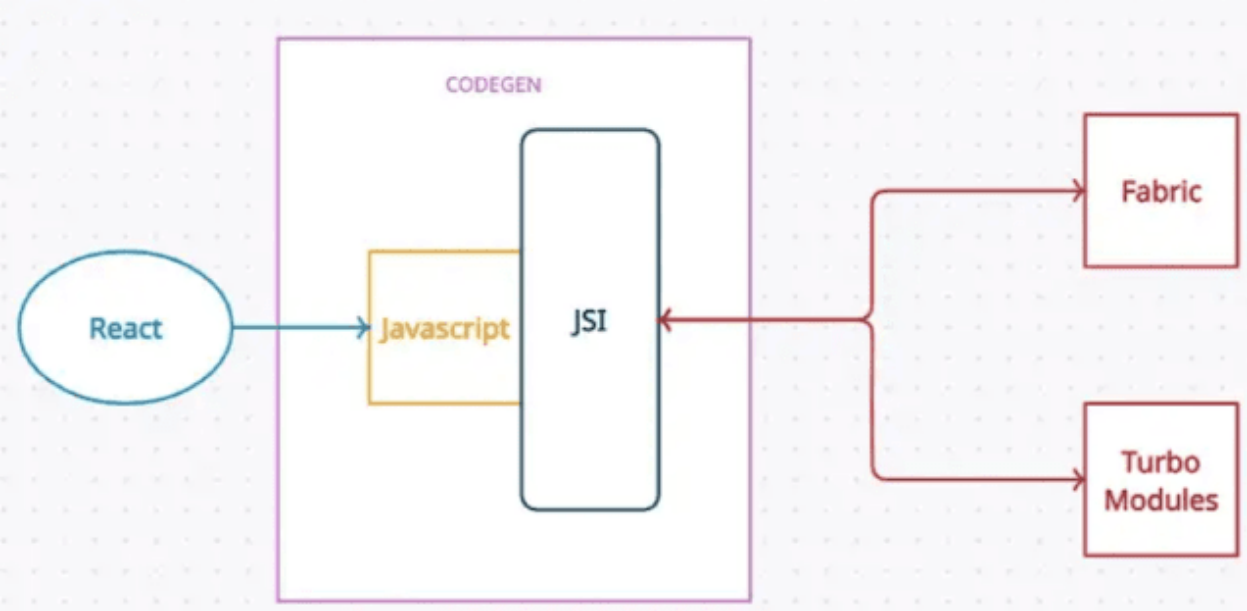
브리지를 통한 JSON 형태의 데이터 전송은 불필요한 데이터 카피가 발생합니다. 비동기식으로 작업이 이루어져서 브리지가 혼잡해지는 치명적인 단점이 있습니다.

또한, 데이터 처리에 있어 강력한 타이핑(typing)을 쓰는게 아닌 느슨한 타이핑으로 문제가 발생할 수도 있습니다.



legacy React Native Architecture

리액트 네이티브의 새로운 아키텍처 : JSI (Javascript Interface)



기존의 아키텍처의 단점을 보완한 새로운 아키텍처를 리액트 네이티브팀이 도입을 하고 있습니다.

JSI(javascript interface) 라는 아키텍처 입니다.

JSI가 가져온 가장 큰 변화는 자바스크립트와 네이티브 파트가 직접 상호 작용한다는 점입니다.

즉, JS와 C++ 간에 인터페이스를 이용해서 상호 간 직접 커뮤니케이션을 한다는 말입니다.

네이티브 쪽 객체를 자바스크립트 콘텍스트 안에서 사용할 수 있다는 점입니다. 이 방법으로 기존 아키텍처의 가장 큰 병목 지점이었던 부분이 JSI로 인해 해결이 되었습니다.

JSI가 상호 작용하는 네이티브 쪽은 두 가지가 있는데, Fabric과 Turbo Modules입니다.

Fabric - C++로 만들어진 렌더링 시스템

Turbo Modules - 필요시에만 로드할 수 있어서 전체적인 앱 반응 속도가 상당히 빨라집니다.

JSI를 가능케 하는 로직 Codegen

Codegen은 javascript와 네이티브 사이드 쪽의 타입 체커를 자동을 해줍니다.

javascript에서 타입을 정확히 지정하면 Codegen에 의해 타입 체커가 되면서 Fabric와 Turbo Modules에 메시지를 전달하는데, 이때 정확한 타입이 전달되어 전에 있었던 느슨한 타이핑이 아닌 강력한 타이핑으로 처리되면서 성능을 향상할 수 있게 됩니다.

++ 새로운 아키텍처에서는 any 타입을 사용할 수 없다고 합니다...

결론적으로 `JSI의 장점` 은

1. 자바스크립트 코드가 네이티브 UI 요소의 레퍼런스를 가질 수 있고 메소드도 직접 호출할 수 있음.
2. 네이티브 코드에 직접 바인딩 되어 속도가 향상됨.
3. JavaScript Core 엔진 말고 다른 엔진도 사용 가능, 실제 Hermes 엔진을 사용하고 있음.
4. 네이티브 모듈을 처음에 모두 다 로드하지 않고 필요할 때 로드하는 방식.
5. 정적 타입 체킹으로 자바스크립트와 네이티브 코드 간 호환성 강화.

현재 진행단계가 어느정도 인지는 모르겠지만 안정화단계를 거치면 네이티브 속도가 상당히 빨라질 것 같습니다..

여담으로 현재 활발하게 개선되고 있는 Flutter를 겨냥하여 성능을 향상시키고 있다는 말도 있습니다.... ㅎ

이상입니다